Table of Contents

Summary

In this spam email dataset, there are 4600 instances with 58 features. First, when processing this dataset, do some explanatory analysis on spam and ham emails. Important findings include spam has longer average word length and contains more words in on email than ham email. Spam email's frequent content include words and characters like "!","free", etc. confusing words. And Spam email's address is often from @email @mail. Therefore, in our daily lives, it is highly recommended that we check the email words and content length before clicking any links in emails, in order to avoid more online fraud.

In this report, the main goal is to build the spam classification filter using the neuron network model. In this case, use the MLP model to build the classification model. The most important thing is to define the input, hidden layer and output layer. Before building the model, I firstly select the most important features to feed into the model. When building this model, first use the basic default parameters, and use the fine-tuning method to find the best parameters, then change the model. After that, the best model, the best accuracy is highly to 0.95.
When compared with other ML models, MLP models have higher accuracy. Neuron Network as a Classification Model has advantages :Flexibility, Adaptation and Scalability.

1.Exploratory Data Analysis

1.1Dataset Dimensions and Summary of the Dataset

- Import pd.read_csv of the spam dataset
- Use "shape","head","info" method to check the basic information of Spam Dataset: There are 4600 instances and 58 attributes,the datatype has no object
- Check there is no missing value

1.2Explanation of the Features

- 4600 instances in this dataset means that there are 4600 emails
- Total is 58 feature columns
- 48 columns of words frequence(word_freq_word), is the frequency count (in percentage) of a particular “WORD” in each email
- 6 columns of characters frequence(char_freq_CH),is the frequency count (in percentage) of a particular character “CH” in each email
- 3 columns of email words information:
    - capital_run_length_average: the average word length in this email
    - capital_run_length_longest: the longest word length in this email
    - capital_run_length_total: how many word counts in this email
- 1 column: spam, means whether it is spam

1.3Attributes Analysis and Insights

1.3.1Email Word Frequency Analysis

In this dataset, we have 4600 emails, including 48 different words frequency columns. Therefore, to make the spam identification more directly, it is important to check the spam emails, in which words count for the higher percentage.
By calculating the average frequency of each word in spam and ham messages, I list the top 10 words with the highest frequency in spam and ham, shown below:

```
Top 10 words with highest frequency in spam messages:
word_freq_you          2.264471
word_freq_your         1.378337
word_freq_will         0.549934
word_freq_free         0.518693
char_freq_!            0.513923
word_freq_our          0.513806
word_freq_all          0.403907
word_freq_mail         0.350879
word_freq_email        0.319182
word_freq_business     0.287547
dtype: float64

Top 10 words with highest frequency in ham messages:
word_freq_you         1.270331
word_freq_george      1.265307
word_freq_hp          0.895448
word_freq_will        0.536316
word_freq_your        0.438693
word_freq_hpl         0.431972
word_freq_re          0.415727
word_freq_edu         0.287120
word_freq_address     0.244436
word_freq_meeting     0.216765
dtype: float64
```

Features of Spam and Ham Emails:

As the results showed, except for the common words “you”, “your”, “will”, the most obvious words in spam are “free”,”all”,”mail”,”email”,”business”.

The most obvious character in spam is “!”. The most obvious words in ham is “you”,”george”, etc.

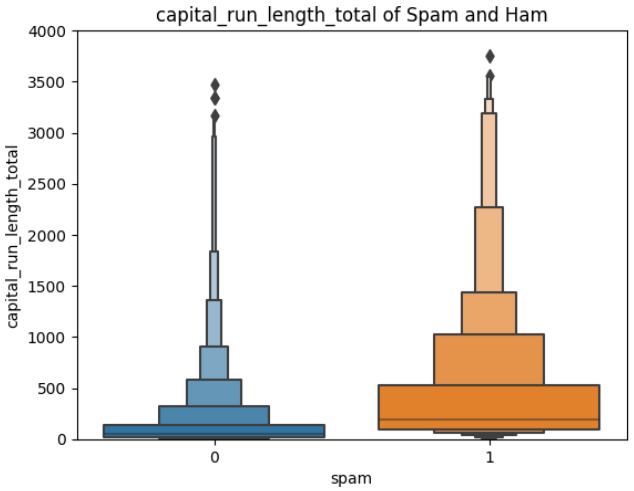
Findings:

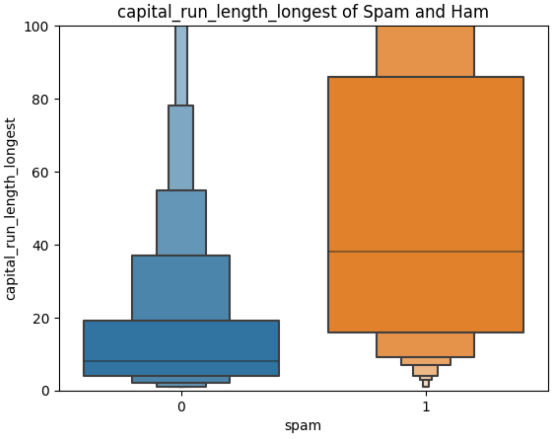
1) the spam email contains more “free”, “!”, which are often used in hyperbolic marketing to attract customers.
2) Spam email often use @email, @mail as address
3) Ham email usually contains the real name like george
4) Ham email often use @edu as reliable address

1.3.2Words Features Analysis

1) Use the boxplot show the difference of average words length, longest words length in spam and ham

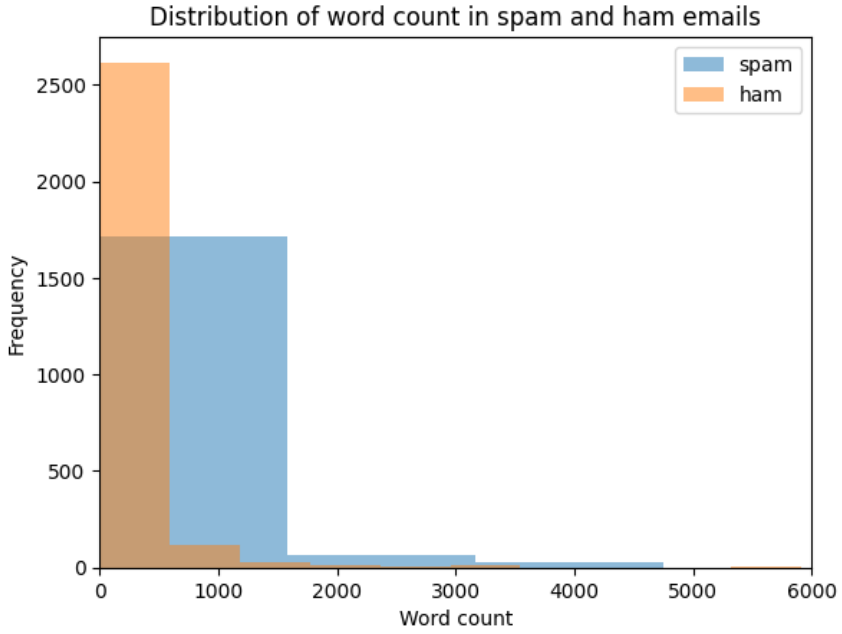
As the figures shown below, Spam emails have longer average_words length than Ham emails, and also Spam have longer longest_words length than Ham Email.





2) Use the hist figure to show the word count of each email of Spam and Ham

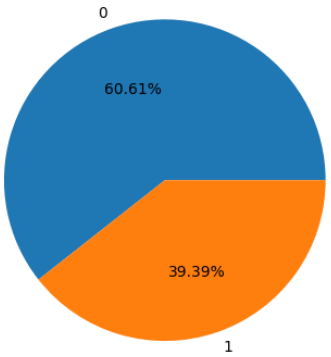
As the figures shown below, Spam emails have more words than Ham emails. Most ham words are between 0-500 words, while spam words are from 0-2000, even there are spam emails words over 5000. Therefore, in daily life, to avoid clicking on spam emails, we can also check the email words. If an email contains too many words, it is likely to be spam.



1.3.3Target Value Analysis

Use the Pie Chart to see the Distribution of Spam and Ham Emails.

As shown below, in this dataset, 60.61% emails are spam emails, 39.39% emails are ham emails.



2.Build the Model: Implementing MLPs with Keras

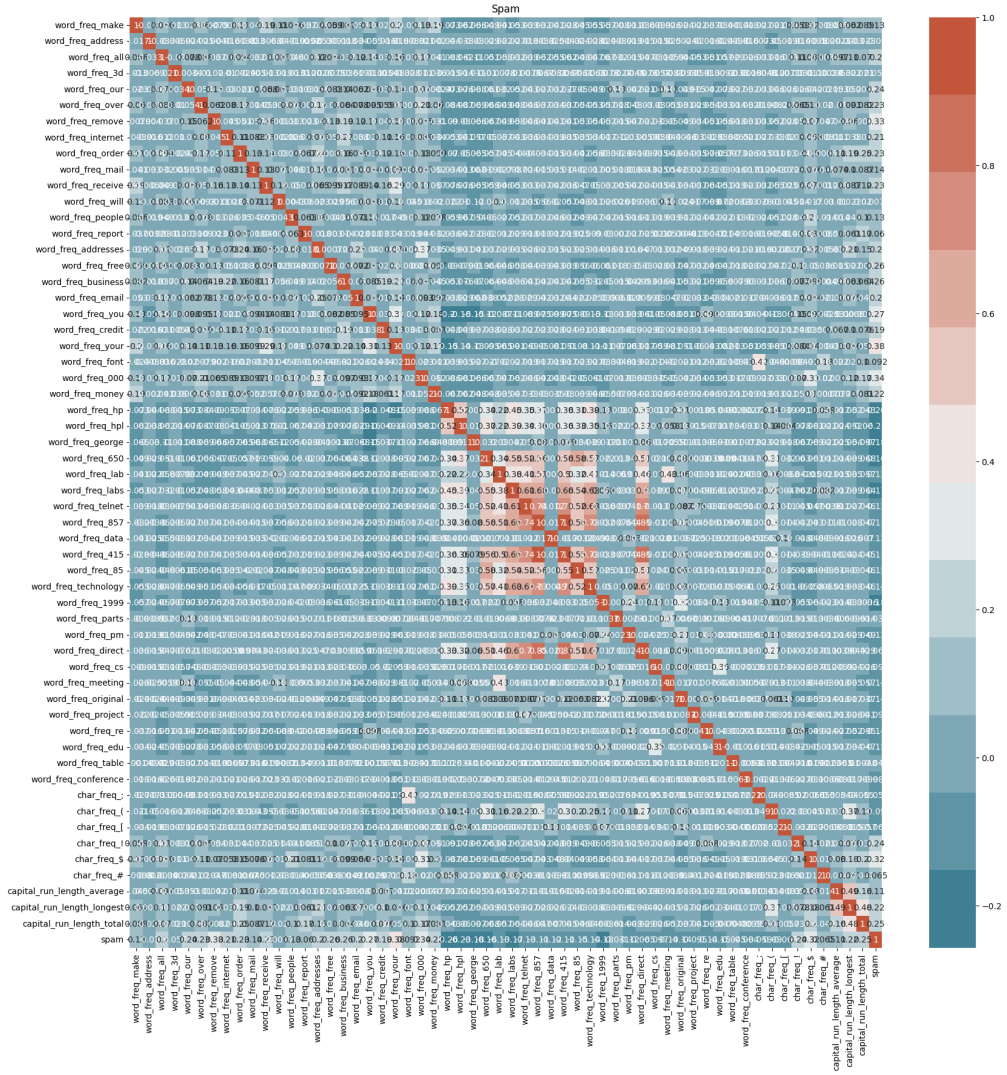
- In this case, I use Multi-Layer Perceptron to build classification model. An MLP has one (passthrough) input layer, one or more hidden layers of TLUs, and one final layer of TLU called the output layer.
- In this Spam and Ham email filter model, the target value only has two values 0 and 1. Therefore, this is a binary classification, only needing a single output neuron using the

logistic activation function: the output will be a number between 0 and 1, which we can interpret as the estimated probability of the positive class. The estimated probability of the negative class is equal to one minus that number.

- As I analyzed before, there are 58 attributes, including 57 input attributes and 1 output attribute.
- Before building the model, I choose to do feature selection, to use the most relevant features to feed into the model.

2.1Feature Selection

First, use the corr() method to calculate the correlations.
As we can see in this model, the 58 features are lengthy and jumped



Therefore, set a benchmark of correlation scores, delete all attributes that are less than absolute value 0.1

```
[15] l=list(df.columns)
     l.pop()
     l
     for i in l:
         corr=df[[i,'spam']].corr()
         a=corr.iloc[1,0]
         if abs(a)<0.1:# set a benchmark of correlation score, to delete <0.1 attributes
             df=df.drop([i],axis=1)
```

In total, the most relevant attributes left are 43

```
df.shape

(4600, 43)
```

2.2Split the Train and Test Dataset

The dataset consists of 4,600 of labelled instances, in .csv format. The dataset is already randomized and the first 3,600 samples are for training, and the remaining 1,000 samples are for testing.

```
[ ] #split into training and testing data
    train_data=df[:3600]
    test_data=df[3600:]
    # Preprocess the data
    X_train_full=train_data.iloc[:,:-1].values
    y_train_full = train_data.iloc[:, -1].values
    X_train,y_train=X_train_full[:-600],y_train_full[:-600]
    X_valid,y_valid=X_train_full[-600:],y_train_full[-600:] # to split the validation data
    X_test = test_data.iloc[:, :-1].values
    y_test = test_data.iloc[:, -1].values
```

X_train Data shape is (3000,42) , X_valid Data shape is (600,42)

2.3Scale the Data Before Building the Model

- Use the StandardScaler to Process all the attributes before feeding the model

```
from sklearn.preprocessing import StandardScaler
# Scale the numerical columns
scaler = StandardScaler()
X_train=scaler.fit_transform(X_train)
X_valid=scaler.fit_transform(X_valid)
X_test=scaler.transform(X_test)
```

2.4Create the model 1 using the Sequential API without fine-tune parameters

- To build a MLP model first, I use the 2 hidden Dense layers first before fine-tuning the parameters

- Assign the model as the Keras Sequential() class
- Specify the input layer using .Flatten() and define the input_shape is 42, because of the feature selection
- The next layer is the first hidden Dense layer with 300 neurons and uses the “relu” activation function
- The next layer is the second hidden Dense layer with 100 neurons and uses the “relu” activation function
- The last layer is the Dense output layer with 10 neurons and uses the “softmax” activation function

```
# Build the model
tf.random.set_seed(42)
model1 = tf.keras.Sequential()
model1.add(tf.keras.layers.InputLayer(input_shape=(42, )))# change the input shape into 42 becaus
model1.add(tf.keras.layers.Flatten())
model1.add(tf.keras.layers.Dense(300, activation="relu"))# the first hidden layer, using relu act
model1.add(tf.keras.layers.Dense(100, activation="relu"))# the second hidden layer,using relu act
model1.add(tf.keras.layers.Dense(10, activation="softmax"))# the output layer, using softmax acti
```

- After building the model, use the compile() method to and specify the *loss function* and the *optimizer* to be used by this model. We can also specify a list of extra metrics to computer during training and evaluation.

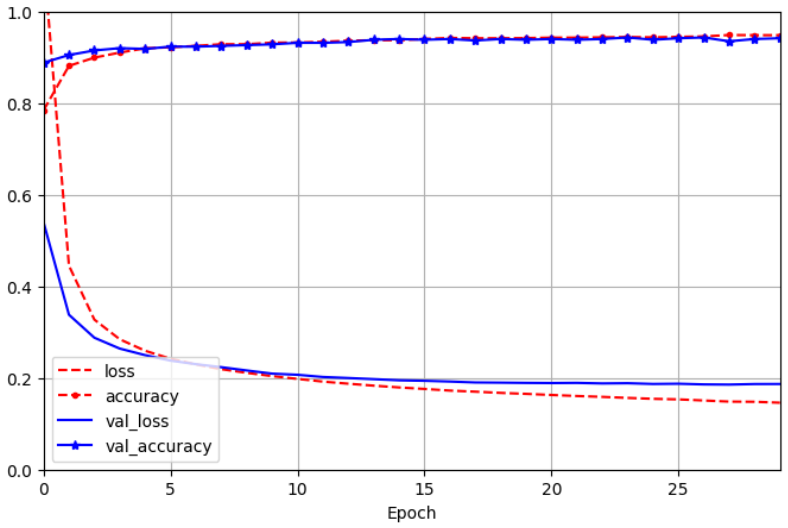
```
model1.compile(loss="sparse_categorical_crossentropy", # because the output
              optimizer="sgd",# use the sgd first
              metrics=["accuracy"])# use the "accuracy" as the metrics
```

  - loss function=”sparse_categorical_crossentropy”
    The classes in this example are exclusive , it is called *sparse labels,* I use loss=“sparse_categorical_crossentropy”
  - optimizer=”sgd”
    Use the “sgd” optimizer first in this case before fine-tuning
  - metrics=[“accuracy”]
    Use the Accuracy as the metrics
- Train the model
  The model is now “compiled” and ready to be trained using .fit() method: epochs are defined to be 30 at first. And also use validation data to specify the validation data set to monitor overfit or underfit
- The training Parameters
  In this first trial, params including “epoches”=30, steps are 94

```
history.params
```

```
{'verbose': 1, 'epochs': 30, 'steps': 94}
```

- The first Learning Curve
  As we can see, in this first model, loss is high



- Model Evaluate the test data is 0.92

```
model1.evaluate(X_test, y_test)
```

```
32/32 [==============================] - 0s 3ms/step - loss: 0.2227 - accuracy: 0.9200
[0.22274565696716309, 0.9200000166893005]
```

2.5Fine tuning the Hyperparameters

- Use the Keras_tuner to fine tune the hyperparameters

```
tf.keras.backend.clear_session()
tf.random.set_seed(42)
import keras_tuner as kt

def build_model(hp):
    n_hidden = hp.Int("n_hidden", min_value=0, max_value=8, default=2)
    n_neurons = hp.Int("n_neurons", min_value=16, max_value=256)
    learning_rate = hp.Float("learning_rate", min_value=1e-4, max_value=1e-2,
                             sampling="log")
    optimizer = hp.Choice("optimizer", values=["sgd", "adam"])
    if optimizer == "sgd":
        optimizer = tf.keras.optimizers.SGD(learning_rate=learning_rate)
    else:
        optimizer = tf.keras.optimizers.Adam(learning_rate=learning_rate)

    model = tf.keras.Sequential()
    model.add(tf.keras.layers.Flatten())
    for _ in range(n_hidden):
        model.add(tf.keras.layers.Dense(n_neurons, activation="relu"))
    model.add(tf.keras.layers.Dense(10, activation="softmax"))
    model.compile(loss="sparse_categorical_crossentropy", optimizer=optimizer,
                  metrics=["accuracy"])
    return model
```

- Build random_search_tuner

```
random_search_tuner = kt.RandomSearch(
    build_model, objective="val_accuracy", max_trials=5, overwrite=True,
    directory="spam", project_name="my_rnd_search", seed=42)
random_search_tuner.search(X_train, y_train, epochs=10,
                           validation_data=(X_valid, y_valid))
```

the result is val_accuracy is 0.63, best val_accuracy so far is 0.94

```
Trial 5 Complete [00h 00m 07s]
val_accuracy: 0.6266666650772095

Best val_accuracy So Far: 0.9449999928474426
Total elapsed time: 00h 00m 56s
```

- Use Top_3 models to get the best parameters
  The best hidden layer is four, the neurons number is 74, the optimizer is adam

```
{'n_hidden': 4,
 'n_neurons': 74,
 'learning_rate': 0.0090512740978246 2,
 'optimizer': 'adam'}
```

2.6Second Model After the Fine-tuning Hyperparameters

- After the fine tuning the hyperparameters, change the hidden layer to 4, and increase the neuron numbers to 74 to build a second model
- Specify the input layer using .Flatten() and define the input_shape is 42, because of the feature selection
- Build four hidden layers
- Change the optimizer parameters to “adam”

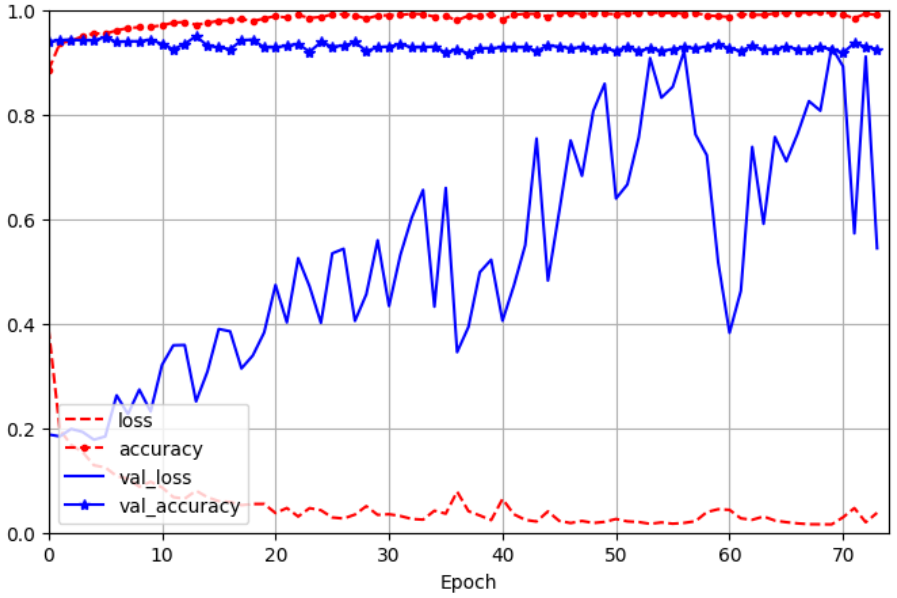
```
model = tf.keras.Sequential()
model.add(tf.keras.layers.InputLayer(input_shape=(42, )))
model.add(tf.keras.layers.Flatten())
model.add(tf.keras.layers.Dense(300, activation="relu"))
model.add(tf.keras.layers.Dense(300, activation="relu"))
model.add(tf.keras.layers.Dense(300, activation="relu"))
model.add(tf.keras.layers.Dense(100, activation="relu"))
model.add(tf.keras.layers.Dense(10, activation="softmax"))
model.compile(loss="sparse_categorical_crossentropy",
              optimizer="adam",
              metrics=["accuracy"])
```

- Train the model

The model is now “compiled” and ready to be trained using .fit() method: epochs are defined to be 74. And also use validation data to specify the validation data set to monitor overfit or underfit

```
history = model.fit(X_train, y_train, epochs=74,
                    validation_data=(X_valid, y_valid))
```

- The second model’s Learning Curve
  As we can see, in this second model, the accuracy is shown below



- Model Evaluate using test data is that accuracy is 0.93

```
32/32 [==============================] - 0s 4ms/step - loss: 0.5773 - accuracy: 0.9290
[0.5773285031318665, 0.9290000200271606]
```

- The best validation accuracy is 0.95 achieved at epoch 14

```
# Find the epoch with the best validation accuracy
best_epoch = history.history['val_accuracy'].index(max(history.history['val_accuracy'])) + 1
best_val_accuracy = max(history.history['val_accuracy'])

print(f"Best validation accuracy of {best_val_accuracy:.4f} achieved at epoch {best_epoch}.")
```

```
Best validation accuracy of 0.9500 achieved at epoch 14.
```

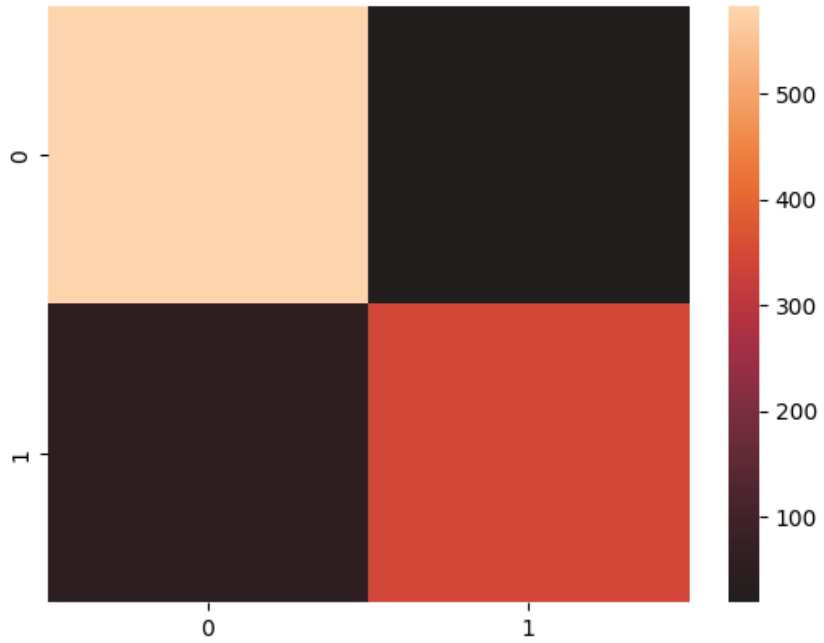
2.7Confusion Matrix

- Use the Confusion Matrix to evaluate the performance of a classification model on new, unseen data
- The cross_val_predict function generates cross-validated predictions for the input features X_test
- The y_test data represents the actual outcomes of the test set, which the model has not seen before

```python
from sklearn.metrics import confusion_matrix
import numpy as np

# Make predictions on the test set
y_pred = model.predict(X_test)
# Convert the predicted probabilities to class labels
y_pred_labels = np.argmax(y_pred, axis=1)
# Compute the confusion matrix
cm = confusion_matrix(y_test, y_pred_labels)
# Print the confusion matrix
print(cm)
```

```
32/32 [==============================] - 0s 2ms/step
[[583  19]
 [ 52 346]]
```



3.Other ML Classification Comparison

- Use the six most common classification models
  the scores of six models are
  LogisticRegression: 0.917
  DecisionTreeClassifier: 0.860
  RandomForestClassifier: 0.936
  SGDClassifier: 0.913
  SVC: 0.907
  KNeighborsClassifier: 0.897

- When compared with these classification models, the accuracy of MLP neuron network is higher.

The advantages of using MLP model than other classification models are:

1) **Flexibility:** Neural networks are flexible and can learn complex nonlinear relationships between input features and output labels, making them well-suited for classification problems with a large number of features or complex data patterns.
2) **Adaptability:** Neural networks can be adapted to a wide range of classification problems by changing the architecture, activation functions, and optimization algorithms, among other hyperparameters.
3) **Scalability:** Neural networks can be scaled up to handle very large datasets by using distributed computing or parallel processing.

4.Conclusion on the MLP SPAM Filter

- From the EDA view, spam email contains longer average word length and contains more words in on email than ham email.
- Spam email's frequent content include words and characters like "!","free", etc. confusing words. And Spam email's address is often from @email @mail. Therefore, in our daily lives, it is highly recommended that we check the email words and content length before clicking any links in emails, in order to avoid more online fraud.
- In this report, the main goal is to build the spam classification filter using the neuron network model. In this case, use the MLP model to build the classification model. The most important thing is to define the input, hidden layer and output layer.
- In this dataset, there are 58 features, at first to do the feature selection, use corr() method to calculate the correlation scores, then define the benchmark of 0.1 to select the most related attributes to build this model.
- At first, set 2 hidden layers, use the "sgd" optimizer, set the epochs=30 to build initial model. However, the result is not so perfect.
- Therefore, use the fine-tune hyperparameters to select the best model's parameters
- Then, build a new model by changing the hidden layer numbers, epochs, optimizer to "adam", the best val_accuracy is improved to 0.95
- When compared with other ML models(LogisticRegression: 0.917, DecisionTreeClassifier: 0.860, RandomForestClassifier: 0.936,SGDClassifier: 0.913,SVC: 0.907, KNeighborsClassifier: 0.897). MLP model has higher accuracy.
- Neuron Network as Classification Model has advantages :Flexibility, Adaptation and Scalability